递归与分治法

本章主题

- 理解递归(Recursion)与分治(Divide and Conquer)的概念
- (本章重点)掌握设计有效的分治策略算法及时间性能分析
  - 时间性能分析是有效分治策略算法设计的依据
- 通过下面的范例学习分治策略设计技巧
  - 二分搜索技术(Binary Search)；
  - 归并排序和快速排序(Merge Sort & Quick Sort)；
  - 大整数乘法；
  - Strassen矩阵乘法；
  - 最接近点对问题（Closest pairwise points)；
  - Convex Hull Finding Problem；
  - 棋盘覆盖；

递归的主要概念

- **递归算法**：直接或间接地调用自身的算法称为递归算法
- **递归函数**：用函数自身给出定义的函数称为递归函数
- **直接/间接递归**：若一个函数直接调用自己，称为直接递归；若函数间接调用自己，则称为间接递归
- **尾递归**：如果一个递归函数中递归调用语句是最后一条执行语句，则称这种递归调用为尾递归
- **递归模型**：一个递归模型是由递归出口(边界条件)和递归体(递归方程)组成。递归出口确定递归到何时结束；递归体确定递归求解时的递推关系。递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

递归式与分治法

凡治众如治寡，分数是也。 ——孙子兵法

释义：指挥大部队战斗与指挥小分队战斗基本原理是一样的，掌握部队建制规模及其相应的名称不同这个特点就行了

分数：部队的编制

- 由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在此情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小至很容易直接求出其解的程度时终止。这自然导致递归过程的产生；
- 分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

直接递归

- 例1. 数的阶乘

阶乘函数可递归地定义为：$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$

边界条件

递归方程

阶乘算法的直接递归版本：

```
int  Factorial(int n)
{
    if (n==1)  return 1;
    else
        return n*Factorial(n-1);
}
```

调用自身

思考：它是尾递归的吗？

递归过程($n = 5$)：

Factorial(5)
{5 * Factorial(4)}
{5 * {4 * Factorial(3)}}
{5 * {4 * {3 * Factorial(2)}}}
{5 * {4 * {3 * {2 * Factorial(1)}}}}
{5 * {4 * {3 * {2 * {1 * Factorial(0)}}}}}
{5 * {4 * {3 * {2 * {1 * 1}}}}}
{5 * {4 * {3 * {2 * 1}}}}
{5 * {4 * {3 * 2}}}
{5 * {4 * 6}}
{5 * 24}
{120}

直接递归 (续)

- 例2. 斐波那契数列（Fibonacci）

无穷数列0，1，1，2，3，5，8，13，21，34，55, …… 被称为斐波那契数列。它可以递归地定义为：

$$f(n)=\begin{cases} n, & n=0,1 \\ f(n-1)+f(n-2), & n\geq 2 \end{cases}$$

边界条件

递归方程

基于此， 斐波那契数列的递归函数可表示为：

```
int Fibonacci(int n)
{
    if (n<=1)
            return n;
    else
            return Factorial(n-1) + Factorial(n-2);
}
```

注意：每一层递归调用的返回结果都依赖于下一层递归调用的返回值,该算法时间复杂度$O(2^n)$

递归优化

- **递归存在的问题：**
  - 时间和空间消耗均较大，存在大量的重复计算，此外每一次函数调用都需要在内存栈中分配空间以保存参数，返回地址以及临时变量。如果递归深度太大，可能导致程序运行时间过长以及**栈溢出**。
- **改进方法：**
  - 尾递归
  - 用户自定义一个栈来模拟系统递归调用工作栈
  - 优化递推关系
  - 求解通项公式
  - 分治策略

尾递归

- **定义：递归调用是整个函数体中最后执行的语句且它的返回值不属于表达式的一部分时。**
- **尾递归函数的特点是在回归过程中不用做任何操作，保证同时只有一个栈帧在栈里存活，节省了大量栈空间。**
  - 仍以阶乘计算为例，尾递归函数增加初始值为1的参数a来维护递归层次的深度，程序结束后a的值即为所求值：

阶乘算法的尾递归版本：

```
int  Factorial(int n, int a=1)
{
    if (n < 0)  return 0;
    else if (n == 0) return 1;
    else if (n == 1) return a;
    else return Factorial(n-1, n*a);
}
```

递归过程($n = 5$)：

Factorial(5)
Factorial(5, 1)
Factorial(4, 5)
Factorial(3, 20)
Factorial(2, 60)
Factorial(1, 120)

递推关系式优化

- 借助**以空间换时间**的思想减少递归算法中存在的大量冗余计算。
- 主要思路：将高冗余的属性加入到一个数组中。用$O(1) \to O(n)$的空间复杂度为代价，换取时间复杂度从$O(2^n) \to O(n)$
  - 仍以**斐波那契数列**为例，使用辅助数组res来记录每次计算的值，以避免重复计算

```
int Fibonacci(int n)
{
    if (n<=1)  return n;
    int res[n+1];
    res[0] = 0;
    res[1] = 1;
    for (int i = 2; i <= n; i++)
        {res[i] = res[i-1]+res[i-2]; }
    return res[n];
}
```

时间复杂度：$O(n)$

空间复杂度：$O(n)$

通项公式

- 斐波那契数列可以找到相应的非递归方式定义：

$$F(n)=\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right)$$

因此我们借助求出的通项公式，仅需$O(1)$的时间即可计算出所求值

- 缺点：公式中引入的无理数在计算机中无法保证计算结果的精度

注意：并非所有问题都可以依靠通项公式求解，这一优化方法使用情况有限

分治法

- 分治法在每层递归上包括三个步骤：
  - Divide(分解)：将原问题划分为若干个子问题
  - Conquer(求解)：递归地解这些子问题；若子问题的规模足够小，则停止递归，直接求解
  - Combine(组合)：将子问题的解组合成原问题的解
- 分治法所能解决的问题一般具有以下几个特征：
  - 该问题的规模缩小到一定的程度就可以容易地解决
  - 该问题可以分解为若干个规模较小的相同问题，即具有最优子结构性质
  - 利用该问题分解出的子问题的解可以合并为该问题的解
  - 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题

分治法求解斐波那契数列

- 定义斐波那契数列第n项为$Fib(n),\ Fib(0) = \mathbf{0},\ Fib(1) = 1$

$$Fib(n) = Fib(n-1) + Fib(n-2)\quad (n >= 2)$$

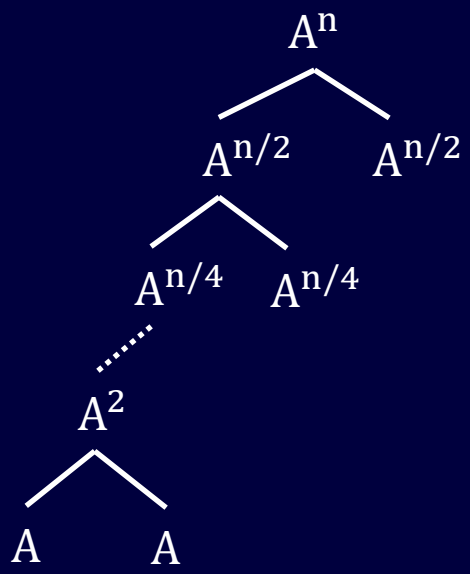
- 将斐波那契数列递推公式写成矩阵相乘形式：

$$\begin{bmatrix} \boldsymbol{Fib(n)} \\ \\ \boldsymbol{Fib(n-1)} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{Fib(n-1)} \\ \\ \boldsymbol{Fib(n-2)} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \\ 1 & 0 \end{bmatrix}^{n-1} \times \begin{bmatrix} \boldsymbol{Fib(1)} \\ \\ \boldsymbol{Fib(0)} \end{bmatrix}$$

不妨设A= $\begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \\ \mathbf{1} & \mathbf{0} \end{bmatrix}$，则$\begin{bmatrix} \boldsymbol{Fib(n)} \\ \\ \boldsymbol{Fib(n-1)} \end{bmatrix} = \mathrm{A}^{\mathrm{n}-1} \times \begin{bmatrix} \boldsymbol{Fib(1)} \\ \\ \boldsymbol{Fib(0)} \end{bmatrix}$，现在，将求解$Fib(n)$转化为求解$\mathrm{A}^{\mathrm{n}-1}$

分治法求解斐波那契数列 (续)

- 问题：如何求解A^n？　　$A^n = A \times A \times \cdots \times A$ (n-1次乘法)
- 方法：为A^n构建一棵二叉树，父节点的值等于两个子节点的乘积，两个子节点的值相同，将下一层得到的值平方返回至上一层



假设树高为k，则有$2^{k-1} = n$，我们只需$\log_2^n$次计算平方即可得到A^n，此外，2阶矩阵相乘本身需要8次乘法，4次加法

时间复杂度：$O(\lg n)$

若n为奇数，先对A^{n-1}进行分治，得到结果再乘以A